Detecting Fraudulent Behavior In E-Commerce

Tina Nguyen and Catherine Le

DESCRIPTION

E-commerce has recently become very popular and continues to rise in popularity. Many people and businesses now rely on it for their transactions, but the increasing risk of fraud comes with its rise. Since there are multiple types of E-commerce fraud, with new ones continuing to arise, we aim to reduce the likelihood of people falling for them by identifying fraudulent behavior.

DATASET DESCRIPTION/ PROCESSING

The dataset we are using is from Kaggle by Shriyash Jagtap[1]. This dataset has 16 features and 23,634 samples. However, this was an excessive amount of features. We decided to run the Scikit mutual_info_classif function on the 16 features to find out which ones have the least impact on the final prediction. Using a mutual information threshold of 0.001, we reduced the list of features to 5.

From		To
Is Fraudulent	Customer Age	Is Fraudulent
Transaction ID	Customer Location	Transaction Amount
Customer ID	Device Used	Account Age Days
Transaction Amount	IP Address	Customer Age
Transaction Date	Shipping Address	Transaction Hour
Payment Method	Billing Address	
Product Category	Account Age Days	
Quantity	Transaction Hour	

Another issue we found was that from the original set of 23,634 samples, only 5% were positive samples. To create a balanced data set, we performed undersampling on the negative samples to develop a 1:1 ratio with the positive samples. Our finalized dataset has five features and 2444 samples.

The original dataset has mixed data types, so we had to preprocess the dataset before running mutual information and undersampling. We used one-hot encoding for categorical features:

[1] https://www.kaggle.com/datasets/shriyashjagtap/fraudulent-e-commerce-transactions

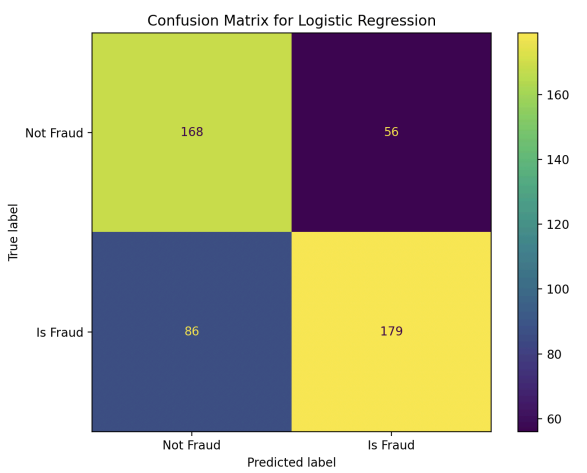
Payment method, product category, customer location, and device used. We used StandardScaler() on numerical features: Transaction Amount, Quantity, Customer Age, Account Age Days, Transaction Hour, Month, and Weekday.

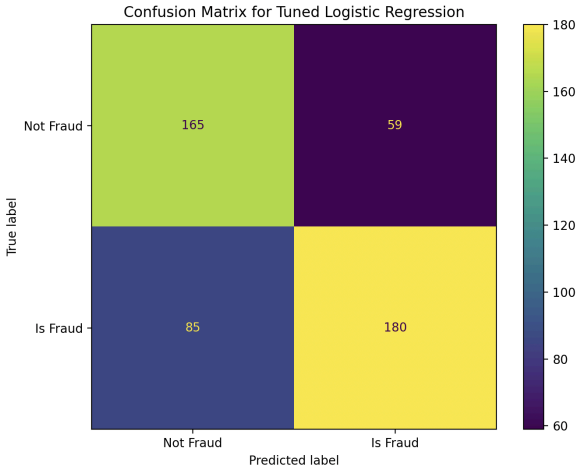
RUNNING EXPERIMENTS

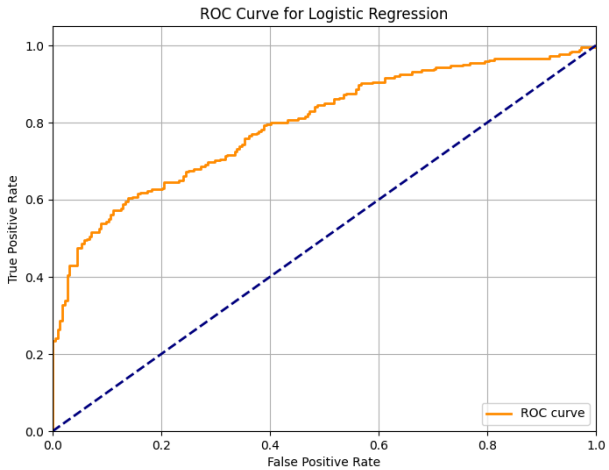
Logistic Regression

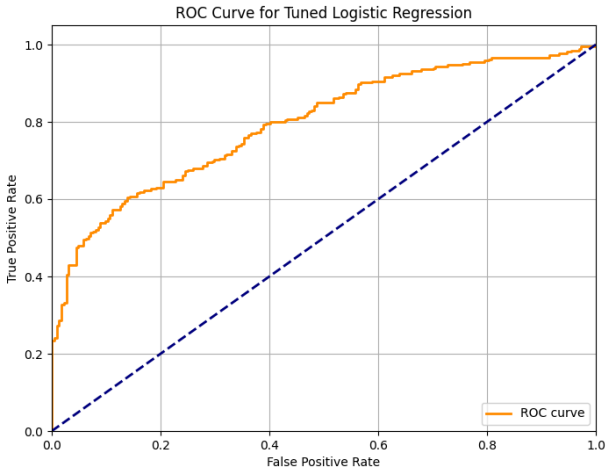
Logistic Regression is a simple linear classifier, so we didn't expect it to perform very well on our data set. However, we still wanted to train it to see what would happen. When we ran our test set through Scikit's default LogisticRegression() model, our results were an accuracy of 0.7096, an AUC of 0.7960, a TPR of 0.6755, and an FPR of 0.2500.

We then tried to hyperparameterize the logistic regression model using GridSearchCV(), optimizing on roc_auc and performing 5-fold cross-validation. We found the hyperparameters of C = 0.1 and max_iter = 100. Our results became an accuracy of 0.7055, an AUC of 0.7956, a TPR of 0.6792, and an FPR of 0.2634. These results show that the logistic regression classifier isn't doing well with our dataset, even with the best parameters.





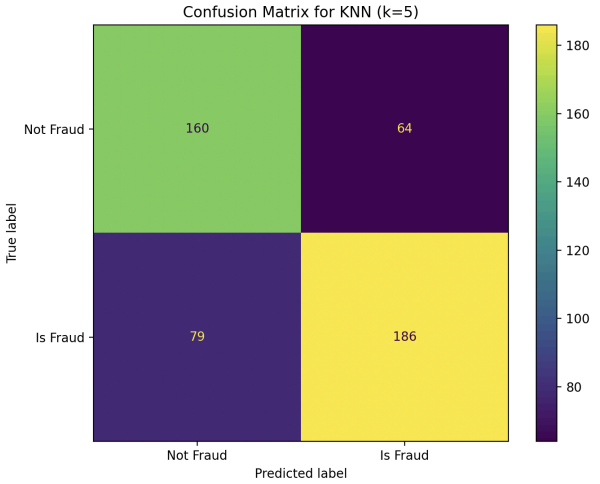


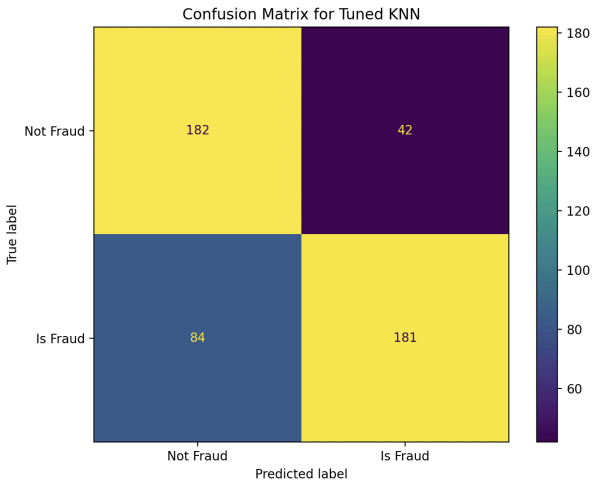


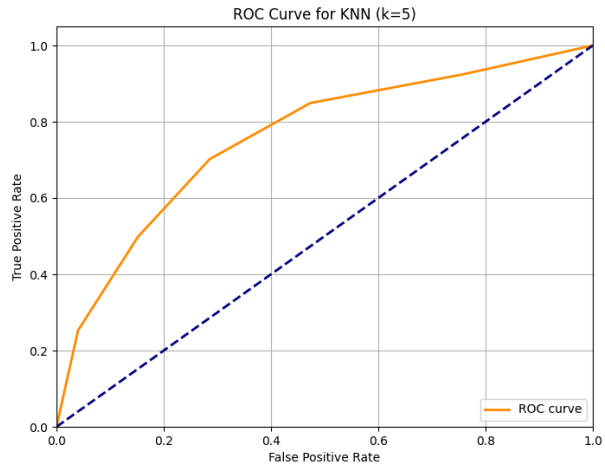
K-Nearest Neighbors

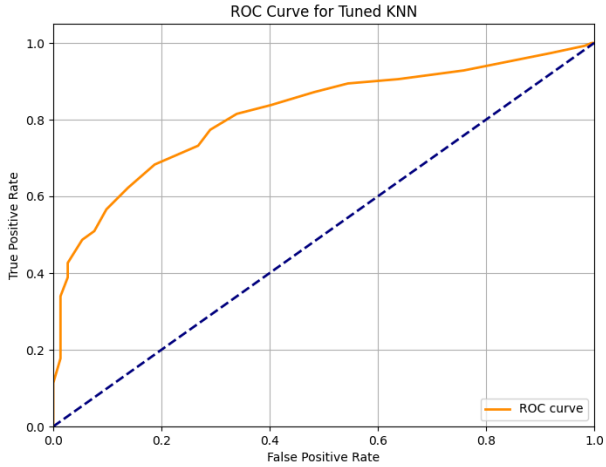
Because logistic regression proved to be a bad classifier for our data because of its linearity, we wanted to try a different classifier that is still simple but non-linear. We decided to experiment with K-Nearest Neighbors. Running the Scikit default KNeighborsClassifier() model with K = 5, we found an accuracy of 0.7076, an AUC of 0.7581, a TPR of 0.7019, and an FPR of 0.2857. This accuracy and AUC score are worse than the default LogisticRegression() results. We predict this to mean that at K = 5, our model is overfitting to the data. We tend to overfit at low K values, and at high K values, we underfit. Following a general rule of thumb that K should stay within a sqrt(N) range, we predict that K should remain below 49. This prediction shows that the default K = 5 is very low, proving that this model is overfitting.

To ensure our prediction is correct, we hyperparametrized the model using GridSearchCV(), again scoring based on roc_auc and doing 5-fold cross-validation, and found the best K value to be 24. Retraining the model with K = 24, we found an accuracy of 0.7423, an AUC of 0.8133, a TPR of 0.6830, and an FPR of 0.1875. The accuracy and AUC scores improved with K = 24 compared to K = 5.





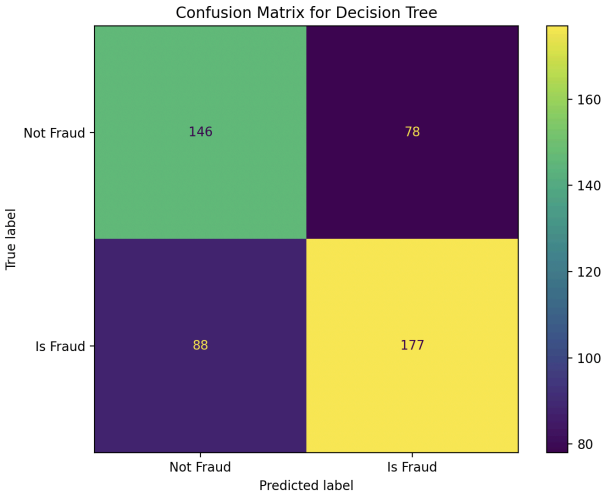


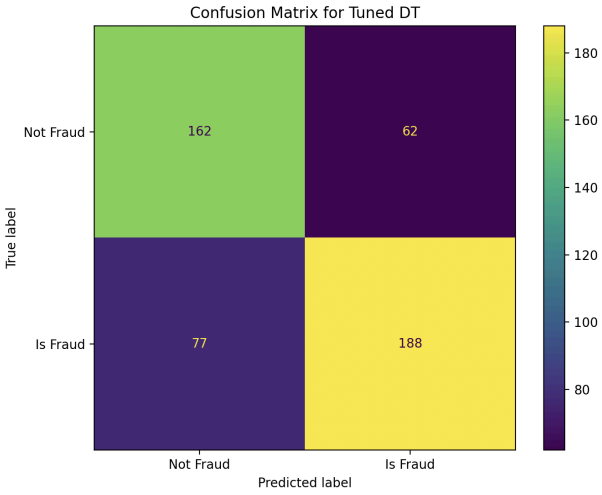


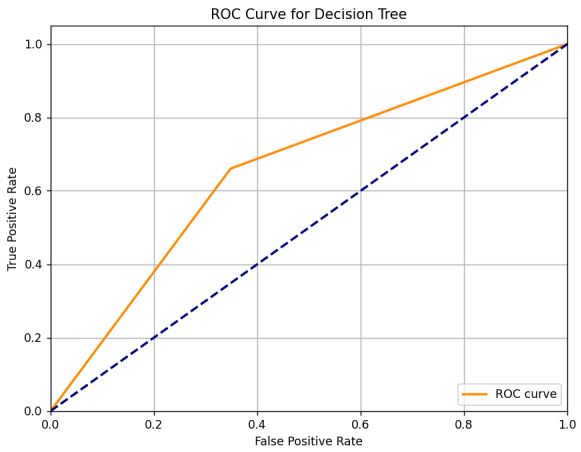
Decision Trees

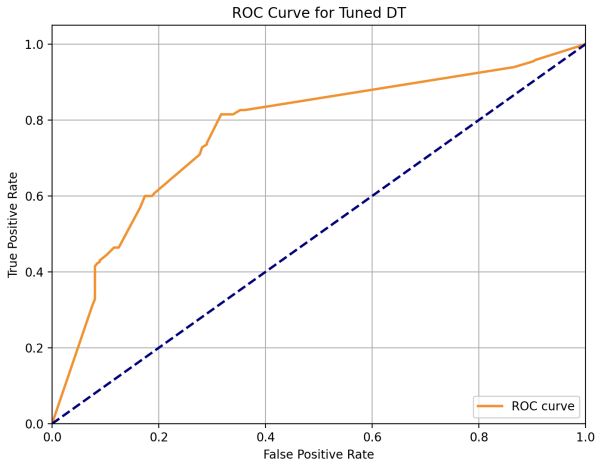
Decision trees are a simple but effective classifier to implement, and they provide a way to visualize our data. Generally, decision trees also work well with mixed data, so we predicted that it might perform well on our dataset. With the Scikit default DecisionTreeClassifier() model, we got an accuracy of 0.6605, an AUC of 0.6592, a TPR of 0.6755, and an FPR of 0.3571; these numbers are comparable to Logistic Regression. Furthermore, since we called the default classifier with no depth limit, we can attribute the poor accuracy to the decision tree overfitting.

To address this, we used GridSearchCV(), scoring on roc_auc and doing 5-fold cross-validation and found that the best parameters are: max_depth = 10, min_sample_leaf = 1, and min_samples_split = 2. These parameters resulted in an accuracy of 0.7157, AUC of 0.7701, TPR of 0.7094, and FPR of 0.2768.





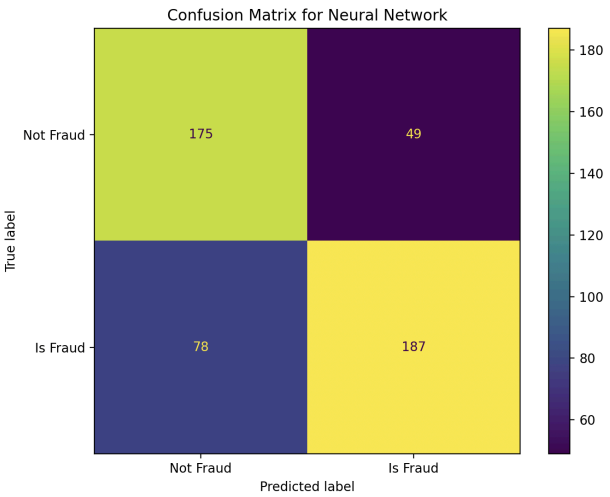


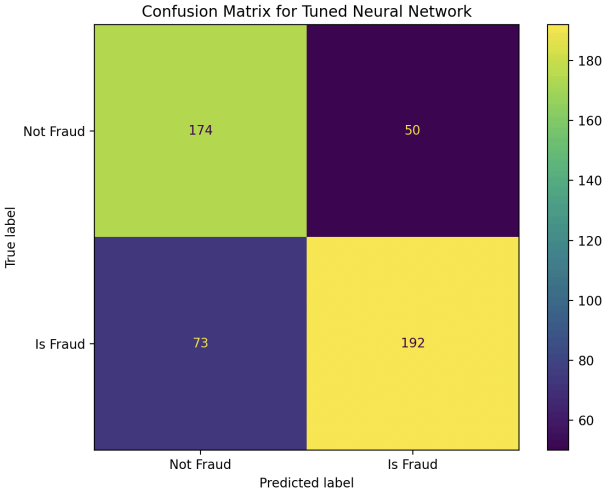


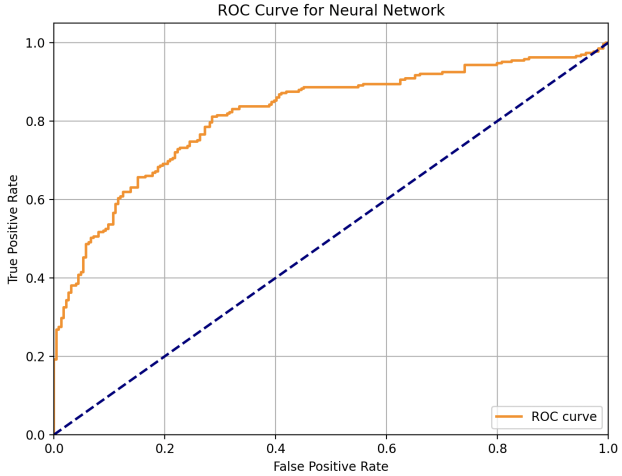
Neural Networks

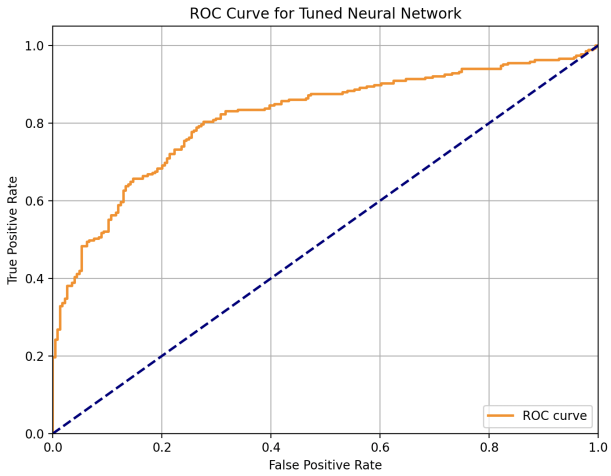
In general, neural networks can solve almost any problem, and we have not experimented with a complex algorithm yet. So we decided to run a neural network on our dataset. Using Scikit's default MLPClassifier(), we got an accuracy score of 0.7444, an AUC of 0.8187, a TPR of 0.7094, and an FPR of 0.2143. Compared to the previous models we've trained, neural networks have had the best default performance so far. This makes sense since it is a more complex model compared to Logistic Regression and K-Nearest Neighbors.

We still wanted to try optimizing the neural network with GridSearchCV(), like the other models. We found that the hyperparameters are: 'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant'. Retraining the model on these parameters resulted in an accuracy of 0.7485, AUC of 0.8152, TPR of 0.7245, and FPR of 0.2232. The results are really close to the default results which means that the default parameters from Scikit already performed well on our dataset, and the hyperparameters didn't have much to improve on.





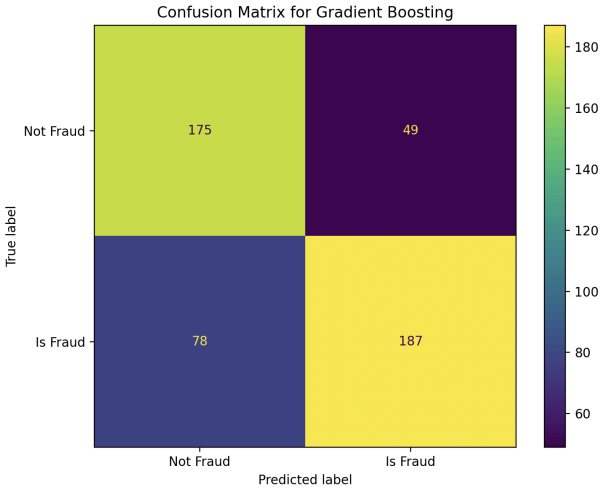


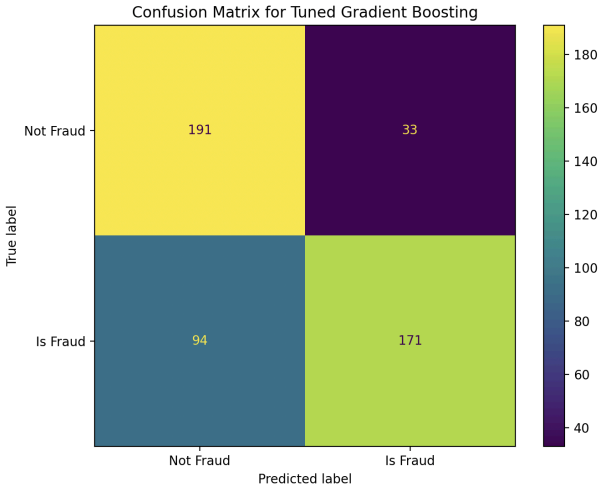


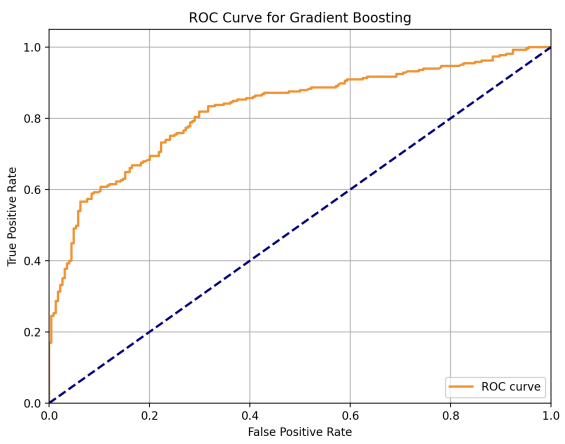
Gradient Boosting

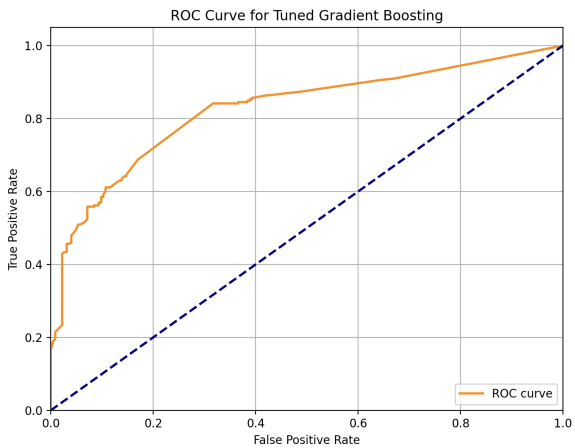
Lastly, we wanted to experiment using an ensemble method on our dataset, so we chose gradient boosting. Gradient boosting is very powerful and effective on data with complex, non-linear relationships, so we predict that this model will give us the best performance. Our dataset is relatively small, so we believed that computationally, gradient boosting wouldn't be too expensive. When running with the default Scikit GradientBoostingClassifier() model, we got an accuracy of 0.7403, an AUC of 0.8239, a TPR of 0.7057, and an FPR of 0.2188. By far, this is the best-performing algorithm based on the AUC score. This was expected, however, we wanted to try optimizing it like the others.

We attempted to optimize our Gradient boosting with GridSearchCV(). In doing so, we found that the best parameters are: learning rate = 0.01, max_depth = 3, max_features = None, n_estiamtors = 100, and subsample = 0.8. With these hyper parameters, we got the results: accuracy = 0.7403, AUC = 0.8251. TPR = 0.6453, and FPR = 0.1473. We observed that Gradient boosting performed the greatest out of all our classifiers– it produced the highest AUC and TPR of all the classifiers, only losing to KNN in terms of accuracy by 0.002.









DISCUSSION

	Accuracy (0.5)	AUC	TPR	FPR
Logistic Regression	0.7055	0.7956	0.6792	0.2634
K-Nearest Neighbors	0.7423	0.8133	0.6830	0.1875
Decision Tree	0.7157	0.7701	0.7094	0.2768
Neural Networks	0.7485	0.8152	0.7245	0.2232
Gradient Boosting	0.7403	0.8251	0.6453	0.1473

The table above has the accumulated results from all five classifiers. From this list, based on the AUC scores, gradient boosting performed the best, then neural networks, then K-nearest neighbors, then logistic regression, then decision tree. The top 3 best performing models made sense; gradient boosting being the top performer was unsurprising given that it's an ensemble method– the iterative boosting/correcting of the decision trees leads to reduced errors. Secondly, neural networks are proficient in handling complex, non-linear relationships in data and can adapt well to the data, contributing to their high performance. Thirdly, KNN is simple, but adapts well to new data, which assists in its performance and ranking. Logistic regression is a linear model, so its better performance than decision tree in terms of AUC was surprising. However, decision trees are prone to overfitting, which likely contributes to its poorer performance.

For all classifiers, we performed GridSearchCV(), focusing on scoring on roc_auc, and performing 5-fold cross-validation. We originally scored on accuracy since our dataset is balanced, however, our accuracy scores were still pretty low even after using the best parameters. We believe that the low accuracy scores were due to the threshold of 0.5 not being optimal, so we decided to recalculate the accuracy scores based on the optimal threshold (max[TPR - FPR]). Retraining the classifiers using the accuracy scores from the optimal thresholds did result in a higher accuracy, however, each classifier had a different optimal threshold. We couldn't use the optimal accuracies to compare all the models because it wouldn't be a fair comparison, so we decided to score all of the classifiers on roc_auc. Roc_auc is threshold-independent so ranking the models against each other using this metric is most fair.

FUTURE PLANS

Given an extra 6 months, we plan to improve our data preprocessing to potentially increase our performance. We believe that the low accuracy may be from our data encoding not being the best since we used one-hot encoding on all features. If we had more time, we would spend more time researching how to properly encode each feature based on its data type. Furthermore, we aim to

implement a wider range of classifiers that may better align with the characteristics of our dataset and yield improved results. We restricted the list of classifiers to the ones we learned in class since these are the ones we knew the most about.